HAND GESTURE CONTROLLED VIDEO GAME

MINOR PROJECT REPORT

Submitted by

Suhas Ganga [RA2111026010281]
Sai Rishyanth Visinigiri [RA2111026010280]
Kavya Reddy Vutukuri [RA2111026010261]
Manisha Manoj [RA2111026010274]

Under the Guidance of

Dr. Kanagaraju P

Assistant Professor, Department of Computational Intelligence

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

with specialization in Artificial Intelligence & Machine Learning

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603203

NOVEMBER 2023

COLLEGE OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR – 603 203
CHENGALPATTU DISTRICT

BONAFIDE CERTIFICATE

Register No **RA2111026010281, RA2111026010280, RA2111026010261, RA2111026010274** certified to be the bonafide work done by **SUHAS GANGA, SAI RISHYANTH VISINIGIRI, KAVYA REDDY VUTUKURI, MANISHA MANOJ** of III Year/V Sem B. Tech Degree Course in **Computer Vision 18CSE390T** in **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**, Kattankulathur during the academic year 2022 – 2023.

FACULTY-IN-CHARGE
Dr. Kanagaraju P
Assistant Professor,
Department of Computational Intelligence
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

HEAD OF THE DEPARTMENT
Dr. R Annie Uthra
Professor and Head,
Department of Computational Intelligence
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

ABSTRACT

The advent of hand gesture control technology has revolutionized the way we interact with digital devices and, in particular, has opened new avenues for immersive gaming experiences. This abstract introduces a comprehensive exploration of a hand gesture-controlled video game system, highlighting its design, implementation, and the impact on the gaming industry.

In recent years, the gaming industry has witnessed a significant shift towards more immersive and intuitive gaming experiences. Hand gesture control, as a non-invasive and natural user interface, has emerged as a promising technology to enhance player engagement. This research delves into the design and development of a hand gesture-controlled video game system, which employs computer vision and machine learning techniques to translate real-time hand gestures into in-game actions. The study also discusses the hardware and software components necessary for the seamless integration of this technology into existing gaming platforms.

The primary focus of this research is to investigate the impact of hand gesture control on the gaming experience. Players can interact with the game environment through natural hand movements, allowing for more immersive and intuitive gameplay. The study also addresses the potential challenges and limitations of this technology, such as gesture recognition accuracy and the need for specialized hardware.

Furthermore, the abstract discusses the implications of hand gesture control for the gaming industry. The integration of this technology into video games not only enhances the overall gaming experience but also has the potential to broaden the demographic of gamers, making gaming more accessible to individuals with varying physical abilities. Additionally, this technology can open up new avenues for innovative game mechanics and design possibilities.

In conclusion, this research explores the exciting possibilities of hand gesture control in video games, offering a glimpse into a future where players can immerse themselves in the gaming world like never before. The findings from this study may pave the way for the continued evolution of gaming technology, ultimately providing more inclusive and engaging experiences for gamers of all backgrounds.

ACKNOWLEDGEMENT

We extend our heartfelt gratitude to the esteemed individuals who have been guiding lights throughout our academic journey and the completion of this course project.

First and foremost, we wish to express our deepest appreciation to our honorable Vice Chancellor, Dr. C. MUTHAMIZHCHELVAN, whose unwavering support has been a constant source of inspiration for all our endeavors.

Our thanks go out to Dr. S. Ponnusamy, our Registrar, for his continuous encouragement, which has motivated us to excel in our academic pursuits.

Dr. T. V. Gopal, Dean of the College of Engineering and Technology, deserves special mention for infusing innovation into every facet of our education and project execution.

We are indebted to Dr. Revathi Venkataraman, Chairperson of the School of Computing, whose mentorship instilled the confidence needed to successfully complete our course project.

Our sincere appreciation goes to our Course Project Faculty, Dr. Kanagaraju P, Assistant Professor in the Department of Computational Intelligence, for his invaluable assistance, timely suggestions, and unwavering guidance throughout the duration of this project.

We also acknowledge the unwavering support of Dr. R. Annie Uthra, Professor and Head of the Department of Computational Intelligence, and our fellow departmental colleagues.

Last but not least, we extend our gratitude to our parents, friends, and all our near and dear ones, whose support, both direct and indirect, contributed significantly to the successful completion of our project.

Above all, we offer our thanks to the Almighty for bestowing His blessings upon us, enabling us to bring our course project to fruition.

I. LIST OF FIGURES

II. LIST OF TABLES

1. INTRODUCTION

1.1 MOTIVATION

Developing a hand gesture-controlled video game is a thrilling and groundbreaking endeavour that carries immense potential for innovation and impact. Here's the motivation that fuels our enthusiasm for this project:

Immersive Gaming Experience: We are driven by the vision of taking gaming to a whole new level. Imagine players fully immersed in the gaming world, using their natural hand movements to interact with the game environment. This project aims to bring that vision to life and create an unparalleled gaming experience.

Inclusivity: We are motivated by the idea of making gaming more inclusive. Hand gesture control can break down barriers, making gaming accessible to individuals of varying physical abilities. This project aligns with our commitment to creating opportunities for everyone to enjoy the world of gaming.

Technological Advancement: The project represents an exciting intersection of cutting-edge technologies, including computer vision and machine learning. We are inspired by the opportunity to push the boundaries of what's possible in the realm of human-computer interaction, exploring the potential of technology to enhance our lives.

Creative Expression: We see this project as a canvas for creative expression. Designing and developing a hand gesture-controlled video game allows us to explore novel game mechanics, challenges, and storytelling methods. It's an opportunity to unleash our creativity and bring a unique gaming experience to the world.

Learning and Growth: The journey of creating this project is a learning adventure. We are motivated by the prospect of acquiring new skills, gaining insights into emerging technologies, and overcoming technical challenges. This project is a catalyst for our personal and professional growth.

Positive Impact: Beyond the realm of entertainment, we are excited about the potential for our hand gesture-controlled video game to have a positive impact. We envision its use in educational settings, therapy, and various fields, contributing to learning, rehabilitation, and more.

Pioneering Spirit: We are fuelled by the desire to be pioneers in this field. Developing a hand gesture-controlled video game is venturing into uncharted territory, and we are eager to leave our mark as trailblazers in the world of interactive entertainment.

Community Engagement: We are motivated by the prospect of building a community around this technology. We see players, developers, and enthusiasts coming together to explore, improve, and expand the possibilities of hand gesture-controlled gaming.

Fun and Enjoyment: At the heart of it all, we are driven by the joy that gaming brings. We believe that our project will not only be a technological achievement but also a source of fun and enjoyment for countless players around the world.

This project is not just a technical endeavour; it's a testament to our passion for gaming, our dedication to innovation, and our commitment to making technology work for the betterment of society. As we embark on this journey to create a hand gesture-controlled video game, we are excited to see our motivation transform into a tangible, immersive, and inclusive gaming experience.

1.2 OBJECTIVE

The central objective of our project is to enhance an existing video game by integrating hand gesture control through computer vision technology. Our overarching goals can be summarized as follows:

Our primary aim is to augment an existing video game with hand gesture control, elevating the gaming experience to new heights. By seamlessly incorporating natural hand gestures, we intend to deepen player immersion and interaction within the game. This objective focuses on the transformation of the existing game into a more engaging and intuitive experience.

In our commitment to innovation and inclusivity, we aim to expand the accessibility of the video game to a wider audience, including individuals with diverse physical abilities. By incorporating hand gesture control, we strive to make the game more inclusive, allowing all players to enjoy and experience the gaming world. Our project is dedicated to the integration of state-of-the-art computer vision technology to accurately interpret and respond to users' hand gestures in real time. This objective underscores our unwavering commitment to technological advancement. By introducing cutting-edge technology into the existing game, we aim to offer players a more dynamic and immersive gaming experience.

Creativity is at the heart of our project. We aspire to introduce innovative game mechanics, challenges, and interactions that are specifically designed for hand gesture control, seamlessly blending with the existing game's storyline and gameplay. Our goal is to reimagine the gaming experience by providing players with fresh and engaging ways to interact with the game world.

Our project offers a unique opportunity for personal and professional growth. Team members have the chance to gain expertise in emerging technologies and software development, particularly in the context of integrating hand gesture control. This objective emphasizes skill development and the acquisition of hands-on experience. Through the collection of user interaction data and feedback, our project aims to contribute to ongoing research in the field of human-computer interaction. The application of hand gesture control in various domains, as demonstrated through the existing game, is an area of interest. By achieving this objective, we hope to provide insights and contribute to the advancement of this technology in different contexts.

Our project underscores the adaptability and versatility of hand gesture control technology. Beyond gaming, we envision its potential application in education, therapy, and professional contexts. This objective highlights the broad range of possibilities for leveraging this technology across diverse domains. Fostering a sense of community among users and developers is a key aim of our project. Collaboration, knowledge sharing, and the creation of a supportive community are central to this objective. By building a network of enthusiasts and professionals, we hope to facilitate growth and innovation in the field of hand gesture-controlled gaming and beyond.

Ultimately, our primary objective is to enhance an existing video game by seamlessly integrating hand gesture control through computer vision technology. We are driven by the desire to offer players an even more enjoyable, immersive, and interactive gaming experience, while contributing to the wider application of this technology in various domains.

1.3 PROBLEM STATEMENT

The challenge we aim to address is the successful integration of hand gesture recognition technology, powered by the OpenCV computer vision library, into the control mechanisms of a video game. The primary goal is to provide players with a more immersive, interactive, and intuitive gaming experience by allowing them to control in-game actions using natural hand gestures. This endeavor involves the intricate process of accurately detecting and interpreting a variety of hand gestures in real-time and seamlessly translating them into game commands, while ensuring a smooth and engaging gaming experience.

1.4 CHALLENGES

- Gesture Accuracy and Recognition: Ensuring precise and reliable recognition of a wide range of hand gestures is a primary challenge. Variations in hand position, lighting conditions, and background clutter must be addressed for accurate gesture detection.
- Latency Minimization: Achieving minimal latency in gesture recognition and translating these gestures into in-game actions is crucial for maintaining a seamless and responsive gaming experience.
- Gesture Diversity: Different games may require various types of hand gestures. Adapting the recognition system to accommodate diverse gestures and gestures specific to the game is a significant challenge.
- User Calibration: Developing a user-friendly and efficient calibration process for individual players to adapt to their unique hand characteristics and gestures is essential to ensure a personalized experience.
- Real-Time Processing: Implementing real-time image processing for gesture recognition can be computationally intensive. Efficient algorithms and hardware acceleration may be needed to handle this challenge.
- Adaptation to Gaming Environments: Ensuring that the system works effectively within the specific gaming environment, including different gaming platforms, screen sizes, and gameplay styles, poses unique challenges.
- User Feedback and Iteration: Establishing a mechanism for users to provide feedback on the gesture recognition system and using that feedback to iteratively improve its accuracy and responsiveness.
- Scalability and Compatibility: Ensuring that the system can be scaled and adapted to work with a variety of video games, gaming consoles, and platforms while maintaining compatibility.
- Design Integration: Seamlessly integrating gesture control into the game's design, ensuring it enhances the gameplay experience rather than feeling like an added feature.
- Inclusivity: Addressing the challenge of making the gesture recognition system inclusive, ensuring it accommodates users with varying levels of familiarity with the technology and individuals with different physical abilities.
- User Education: Providing resources and guidance to help users learn and effectively use hand gestures for game control, especially for those who may be new to this method.

2. LITERATURE SURVEY

Steven Yap, Billy Nicholas Panggiri, Garry Darian, Yohan Muliono, Simeon Yuda Prasetyo, "Enhancing BISINDO Recognition Accuracy Through Comparative Analysis of Three CNN Architecture Models", 2023 International Conference on Information Management and Technology (ICIMTech), pp.732-737, 2023 [1].

This paper aims to develop an effective and efficient system for hand gesture recognition of Indonesian Sign Language (BISINDO) using a comparative analysis of three CNN architectures. The Python programming language is used as the development environment, together with the free edition of Google Colab. Google Colab provides a free online platform for coding and experimentation. PyTorch is used as a framework for training models, exploiting its Python programming language capabilities.

Shengdang He, Yuanyuan Zou, Derui Wang, "A Review of Vision-Based Hand Action Recognition Techniques", 2023 42nd Chinese Control Conference (CCC), pp.8413-8418, 2023 [2].

In recent years, with the development of artificial intelligence, hand action recognition has been widely used in human-computer interaction and virtual reality, etc. To help researchers understand the current development status of vision-based hand action recognition technology, this paper presents the latest progress in hand action recognition technology. The methods are divided into machine learning-based hand action recognition and deep learning-based hand action recognition.

M. Karam, A framework for research and design of gesture-based human-computer interactions [D], University of Southampton, 2006 [3].

In this work, we present a theoretical framework to support a systematic approach to researching and designing gesture-based interactions. We propose four categories —physical gestures, input devices, output technologies, and user goals —as the basis from which the framework extends. Each category is defined in terms of manipulatable parameters, and their affect on the user experience. Parameters can be tested using empirical experiments, and amended using qualitative methods. The framework is intended for use as a tool to guide research and design, and presents a structure for providing a theoretical understanding of gesture interactions.

3. REQUIREMENT ANALYSIS

Requirement analysis for a project involving hand gesture recognition for controlling games is a critical phase in the project's development. This process helps define the project's scope, objectives, and constraints. Below is a comprehensive requirement analysis for such a project:

Project Scope:

Define the project's overall scope, including the specific goals and objectives.

Target Audience:

Identify the intended audience for the project. Is it designed for casual gamers, professional gamers, or individuals with disabilities?

Hardware and Software Requirements:

Specify the hardware components required for gesture recognition (e.g., depth cameras, RGB cameras, sensors). List the software tools, libraries, and frameworks necessary for developing and implementing gesture recognition algorithms.

Gesture Recognition Algorithms:

Detail the specific algorithms or machine learning techniques that will be used for hand gesture recognition (e.g., deep learning, computer vision).

Real-time Processing:

Determine if real-time gesture recognition is a requirement for the project and specify any latency constraints.

Gesture Dataset:

Identify the dataset(s) that will be used for training and testing the gesture recognition system.

Game Platform Integration:

Specify the gaming platforms the project will support (e.g., PC, consoles, mobile devices).

Usability and User Experience:

Define user experience (UX) and usability requirements, considering factors like intuitiveness, ease of use, and accessibility.

Testing and Validation:

Specify the testing methodologies and criteria for validating the accuracy and reliability of the gesture recognition system.

List of commonly used libraries and frameworks for different components in the project

Computer Vision and Image Processing:

- OpenCV (Open Source Computer Vision Library): A widely used open-source library for computer vision and image processing tasks.
- Dlib: Known for its face detection and facial landmark recognition, Dlib also provides tools for object tracking and hand gesture recognition.

Deep Learning and Machine Learning:

- TensorFlow: An open-source deep learning framework developed by Google for building and training machine learning models.
- PyTorch: A popular open-source deep learning framework developed by Facebook's AI Research lab.
- Keras: A high-level neural networks API that runs on top of TensorFlow or other deep learning frameworks.
- 3D Gesture Recognition:

Machine Learning Models and Pre-trained Models:

- MobileNet: A family of small, efficient neural network architectures suitable for mobile and embedded vision applications.
- YOLO (You Only Look Once): A real-time object detection system that can be used for hand detection.
- EfficientNet: A family of convolutional neural networks that offer a good balance between accuracy and computational efficiency.

Data Preprocessing and Analysis:

- NumPy: A fundamental package for scientific computing with Python.
- Pandas: A library for data manipulation and analysis.
- Scikit-learn: A machine learning library for Python that provides simple and efficient tools for data mining and data analysis.

4. ARCHITECTURE AND DESIGN

MediaPipe is a cross-platform pipeline framework to build custom machine learning solutions for live and streaming media. The framework was open-sourced by Google and is currently in the alpha stage.

In computer vision pipelines, those components include model inference, media processing algorithms, data transformations, etc. Sensory data such as video streams enter the graph, and perceived descriptions such as object-localization or face-keypoint streams exit the graph.

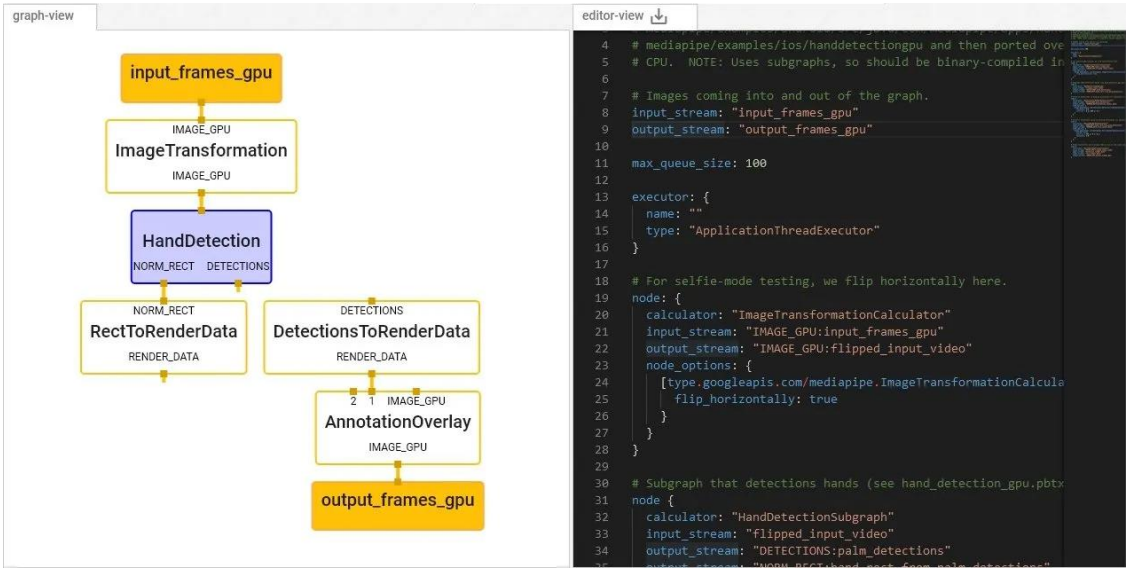


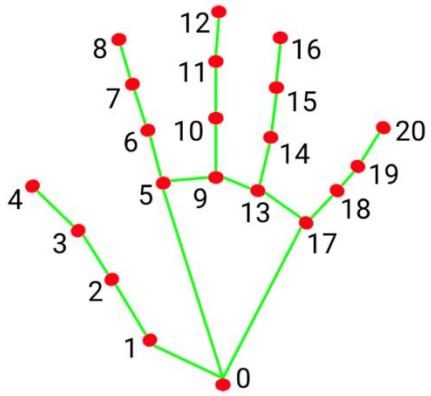
Figure 4.1: MediaPipe Hand Detection Demo using the Visualizer Tool

MediaPipe Architecture

MediaPipe allows a developer to prototype a pipeline incrementally. A vision pipeline is defined as a directed graph of components, where each component is a node ("Calculator"). In the graph, the calculators are connected by data "Streams." Each stream represents a time-series of data "Packets." Together, the calculators and streams define a data-flow graph. The packets which flow across the graph are collated by their timestamps within the time-series. Each input stream maintains its own queue to allow the receiving node to consume the packets at its own pace. The pipeline can be refined incrementally by inserting or replacing calculators anywhere in the graph. Developers can also define custom calculators. While the graph executes calculators in parallel, each calculator executes on at most one thread at a time. This constraint ensures that the custom calculators can be defined without specialized expertise in multithreaded programming.

Hand Land Marker Model

The hand landmark model bundle detects the key point localization of 21 hand-knuckle coordinates within the detected hand regions. The model was trained on approximately 30K real-world images, as well as several rendered synthetic hand models imposed over various backgrounds.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 4.2: Hand Land Marker MediaPipe

The hand landmarker model bundle contains a palm detection model and a hand landmarks detection model. The Palm detection model locates hands within the input image, and the hand landmarks detection model identifies specific hand landmarks on the cropped hand image defined by the palm detection model.

Since running the palm detection model is time consuming, when in video or live stream running mode, Hand Landmarker uses the bounding box defined by the hand landmarks model in one frame to localize the region of hands for subsequent frames. Hand Landmarker only re-triggers the palm detection model if the hand landmarks model no longer identifies the presence of hands or fails to track the hands within the frame. This reduces the number of times Hand Landmarker tiggers the palm detection model.

There were four phased that is used to develop this project. The phased includes analysis, concept design, implementation and evaluation.

A. Analysis Phase

In this game the car will be controlled by hand gestures. The hand gesture will be playing the role of keystrokes. The car will be controlled by multiple orientations of the hand. The orientation is tracked by the hand landmark. There are 21 landmarks that have been detected to track the correct position of the hand and make a meaningful gesture by hand movement.

B. Concept Design Phase

In the game, the player will have a car. The car can be controlled by the keyboard stroke. The car will face some obstacles like another car. The player has to control the car to survive and make a score. But additional features of hand gesture controlling has been added. The car can be moved right and left by the keyboard. The movement can be controlled by the right and left gestures. The index finger of both fingers will be detected and the system will take the decision which is the gesture of moving right and which is the gesture of the moving left. The gesture will be captured by the camera attached to the device. This game is applicable to the webcam. The webcam will capture the gesture of the player's hands and send it to the system control. The angle between the figure will be calculated to taking the right decision for the system.

C. Implementation Phase

In this game, three Landmarks has been used 8, 5, and 0, i.e., INDEX_FINGER_TIP, INDEX_FINGER_MCP, and WRIST, from 21 landmarks respectively. Python programming language was used in this project. Next, the angle between these landmarks were calculated and based on that angle the directions or the inputs were detected. The libraries that were used are mediapipe, cv2, numpy, uuid, os, pynput.keyboard. The mediapipe will direct the hand form the webcam input and point the 21 landmark of the hand. "cv2" is being used for the real time right-left showing in the monitor. The matrix calculation is being done by the numpy library. "uuid" library is used for generating the unique ids. "Os" will help to communicate with the operating system. The keyboard strokes will be controlled through pynput.keyboard.

D. Evaluation Phase

The left-right command is directed by the hand gesture. In the output screen, the left-right command will be shown. This command is equivalent to the keyboard And then the car can move left and right.

5. IMPLEMENTATION

Function Name	Description	Value Range	Default Value
running_mode	Sets the running mode for the task. There are three modes: IMAGE: The mode for single image inputs. VIDEO: The mode for decoded frames of a video. LIVE_STREAM: The mode for a livestream of input data, such as from a camera. In this mode, resultListener must be called to set up a listener to receive results asynchronously.	{IMAGE, VIDEO, LIVE_STREAM}	IMAGE
num_hands	The maximum number of hands detected by the Hand landmark detector.	Any integer > 0	1
min_hand_detection_confidence	The minimum confidence score for the hand detection to be considered successful in palm detection model.	0.0 - 1.0	0.5
min_hand_presence_confidence	The minimum confidence score for the hand presence score in the hand landmark detection model. In Video mode and Live stream mode, if the hand presence confidence score from the hand landmark model is below this threshold, Hand Landmarker triggers the palm detection model. Otherwise, a lightweight hand tracking algorithm determines the location of the hand(s) for subsequent landmark detections.	0.0 - 1.0	0.5

min_tracking_confidence	The minimum confidence score for the hand tracking to be considered successful. This is the bounding box IoU threshold between hands in the current frame and the last frame. In Video mode and Stream mode of Hand Landmarker, if the tracking fails, Hand Landmarker triggers hand detection. Otherwise, it skips the hand detection.	0.0 - 1.0	0.5
result_callback	Sets the result listener to receive the detection results asynchronously when the hand landmarker is in live stream mode. Only applicable when running mode is set to LIVE_STREAM	N/A	N/A

Table: 5.1 Functions Utilized in Hand Gesture Recognition

Below is the step-wise approach for Hand landmarks detection

STEP-1: Import all the necessary libraries, In our case only two libraries are required.

```
In [ ]: # Import Libraries
        import cv2
        import time
        import mediapipe as mp
```

Figure 5.1: Necessary Libraries

STEP-2: Initializing Holistic model and drawing utils for detecting and drawing landmarks on the image.

```
In [ ]: # Grabbing the Holistic Model from Mediapipe and
        # Initializing the Model
        mp_holistic = mp.solutions.holistic
        holistic_model = mp_holistic.Holistic(
            min_detection_confidence=0.5,
            min_tracking_confidence=0.5
        )

        # Initializing the drawing utils for drawing the facial landmarks on image
        mp_drawing = mp.solutions.drawing_utils
```

Figure 5.2: Holistic Model

STEP-3: Detecting Hand landmarks from the image. Holistic model processes the image and produces landmarks for Face, Left Hand, Right Hand and also detects the Pose of the capture the frames continuously from the camera using OpenCV.

- Convert the BGR image to an RGB image and make predictions using initialized holistic model.
- The predictions made by the holistic model are saved in the results variable from which we can access the landmarks using results.right_hand_landmarks, results.left_hand_landmarks respectively.
- Draw the detected landmarks on the image using the draw_landmarks function from drawing utils.
- Display the resulting Image.

The holistic model produces 468 Face landmarks, 21 Left-Hand landmarks, and 21 Right-Hand landmarks. The individual landmarks can be accessed by specifying the index of the required landmark.

In [ ]:
```
# (0) in VideoCapture is used to connect to your computer's default camera
capture = cv2.VideoCapture(0)

# Initializing current time and precious time for calculating the FPS
previousTime = 0
currentTime = 0

while capture.isOpened():
    # capture frame by frame
    ret, frame = capture.read()

    # resizing the frame for better view
    frame = cv2.resize(frame, (800, 600))

    # Converting the from BGR to RGB
    image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Making predictions using holistic model
    # To improve performance, optionally mark the image as not writeable to
    # pass by reference.
    image.flags.writeable = False
    results = holistic_model.process(image)
    image.flags.writeable = True

    # Converting back the RGB image to BGR
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    # Drawing the Facial Landmarks
    mp_drawing.draw_landmarks(
    image,
    results.face_landmarks,
    mp_holistic.FACEMESH_CONTOURS,
    mp_drawing.DrawingSpec(
    color=(255,0,255),
    thickness=1,
    circle_radius=1
    ),
```

```
    mp_drawing.DrawingSpec(
    color=(0,255,255),
    thickness=1,
    circle_radius=1
   )
   )

    # Drawing Right hand Land Marks
    mp_drawing.draw_landmarks(
    image,
    results.right_hand_landmarks,
    mp_holistic.HAND_CONNECTIONS
    )

    # Drawing Left hand Land Marks
    mp_drawing.draw_landmarks(
    image,
    results.left_hand_landmarks,
    mp_holistic.HAND_CONNECTIONS
    )

    # Calculating the FPS
    currentTime = time.time()
    fps = 1 / (currentTime-previousTime)
    previousTime = currentTime

    # Displaying FPS on the image
    cv2.putText(image, str(int(fps))+" FPS", (10, 70), cv2.FONT_HERSHEY_COMPLEX, 1, (0,255,0), 2)

    # Display the resulting image
    cv2.imshow("Facial and Hand Landmarks", image)

    # Enter key 'q' to break the loop
    if cv2.waitKey(5) & 0xFF == ord('q'):
        break

# When all the process is done
# Release the capture and destroy all windows
capture.release()
cv2.destroyAllWindows()
```

Figure 5.3: Detecting Hand Landmarkers

Get the index of all the individual landmarks:

```
In [ ]: # Code to access landmarks
        for landmark in mp_holistic.HandLandmark:
            print(landmark, landmark.value)

        print(mp_holistic.HandLandmark.WRIST.value)
```

Figure 5.4: Index of the Individual Landmarks

Output:

```
HandLandmark.WRIST 0
HandLandmark.THUMB_CMC 1
HandLandmark.THUMB_MCP 2
HandLandmark.THUMB_IP 3
HandLandmark.THUMB_TIP 4
HandLandmark.INDEX_FINGER_MCP 5
HandLandmark.INDEX_FINGER_PIP 6
HandLandmark.INDEX_FINGER_DIP 7
HandLandmark.INDEX_FINGER_TIP 8
HandLandmark.MIDDLE_FINGER_MCP 9
HandLandmark.MIDDLE_FINGER_PIP 10
HandLandmark.MIDDLE_FINGER_DIP 11
HandLandmark.MIDDLE_FINGER_TIP 12
HandLandmark.RING_FINGER_MCP 13
HandLandmark.RING_FINGER_PIP 14
HandLandmark.RING_FINGER_DIP 15
HandLandmark.RING_FINGER_TIP 16
HandLandmark.PINKY_MCP 17
HandLandmark.PINKY_PIP 18
HandLandmark.PINKY_DIP 19
HandLandmark.PINKY_TIP 20
0
```

Figure 5.5 Output for Hand Landmarks with Index

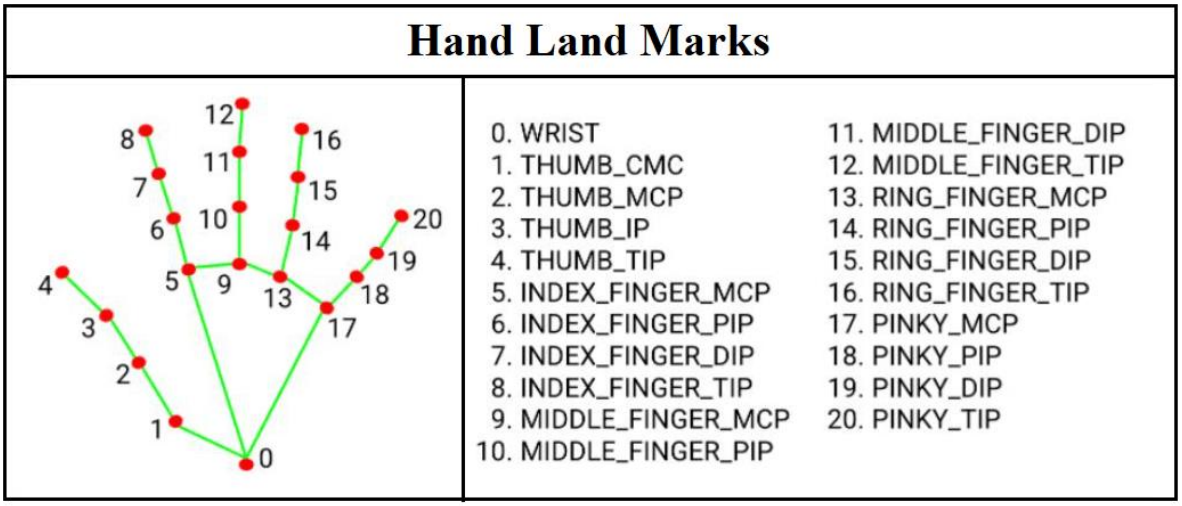


Figure 5.6 Hand Landmarks and their Indices

Final Output:

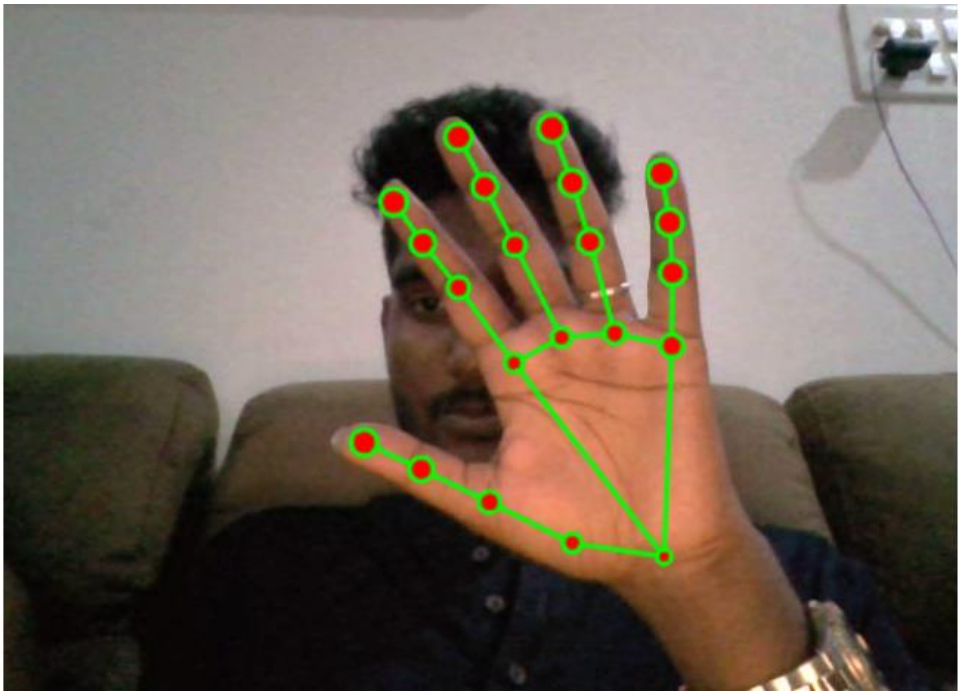


Figure 5.7 Hand Landmark Detection Model

STEP 4: Create Graphics Objects and elements were represented in 2D graphics. The score is shown but there are no visual effects.

STEP 5: Environment Camera is needed for the project or game to work. Pycharm was used for this Python development.

6. RESULT AND ANALYSIS

User Acceptance Testing

User acceptance testing (UAT) is mainly a black box technique used to analyse something by comparing factors like the expected results and actual results. Going thoroughly to the table helps to improve software quality and overlook the issue regarding the environmental factors in the real world . Here, UAT was mainly focused on the core and critical components of the system.

Event	Expected Results	Actual Results
Detect Hand Gesture	Hand gesture will be recognized and work smoothly	Hand gesture recognition is working fine
Character Movement	Swiping hand towards a direction will work as a command to move the character	The character movement is working perfectly and precisely
Game Mechanics	Game will start and will end when the character crashes with objects	There is no start menu and the game restart automatically after crush

Table 6.1: Gesture Testing Results

Based on the table, it shows that the hand detection and car movement are working fine. However, the game mechanics need some adjustment and improvement.

Figure 6.1: Demonstration of Hand Gesture Controlled Subway Surfers Game

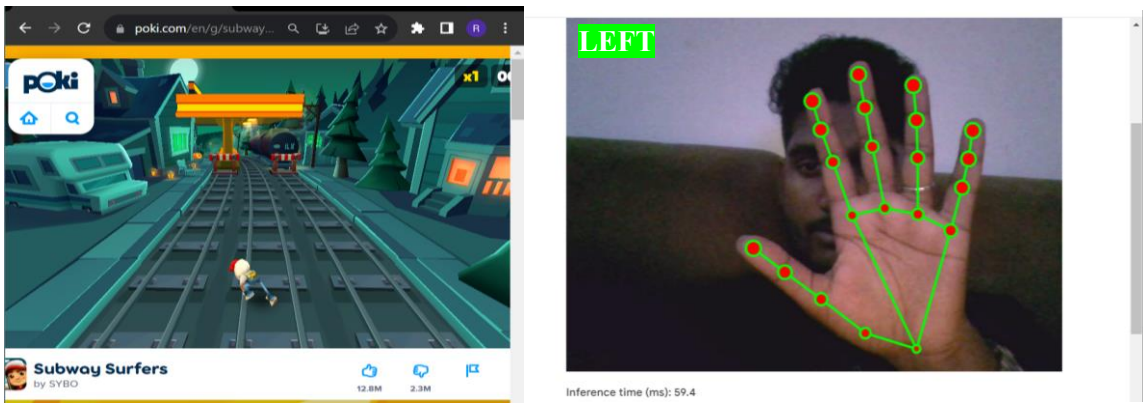


Figure 6.2: Left Hand Gesture to Turn Left

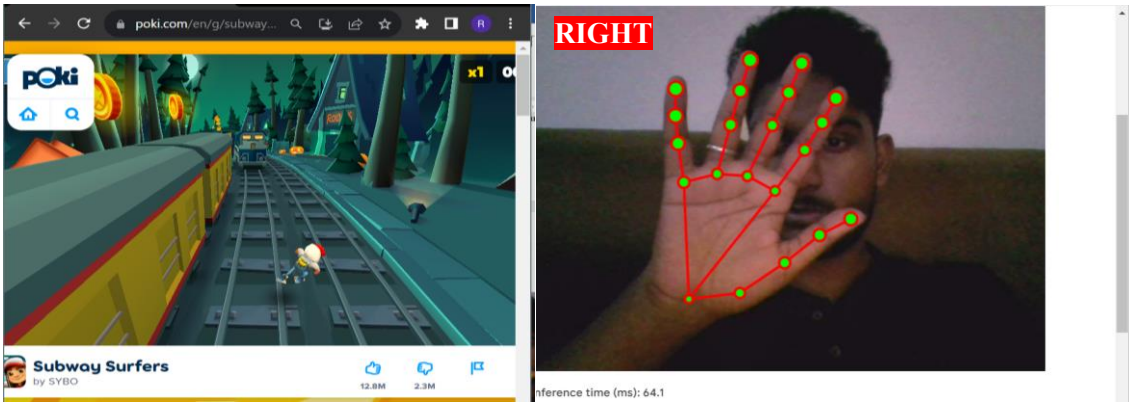


Figure 6.3: Right Hand Gesture to Turn Right

7. CONCLUSION

Following the implementation and thorough testing of Subway Surfers with hand gesture control, it is evident that this novel approach has enhanced the gaming experience significantly. The introduction of hand gesture control has not only added an element of excitement to the game but has also improved the overall gameplay quality.

In the course of developing Subway Surfers with gesture recognition, it is essential to acknowledge some limitations. While the gesture-controlled version of the game is entertaining, there are certain visual elements, like sound effects and user interfaces, that could be further improved and integrated for a more immersive gaming experience. It's important to note that Subway Surfers was chosen as the initial game for its simplicity and minimalistic controls, making it an ideal starting point.

The methodology applied to Subway Surfers can be extended to more complex and 3D games, albeit with potential challenges in maintaining efficiency and accuracy. Advanced data training and gathering techniques can be employed to enhance the method's precision, making it suitable for more intricate gaming scenarios.

In summary, the integration of hand gesture control into Subway Surfers has proven to be a promising innovation that not only augments the gaming experience but also opens up opportunities for future enhancements and refinements, making the game even more captivating and engaging.

8. REFERENCES

[1] Steven Yap, Billy Nicholas Panggiri, Garry Darian, Yohan Muliono, Simeon Yuda Prasetyo, "Enhancing BISINDO Recognition Accuracy Through Comparative Analysis of Three CNN Architecture Models", 2023 International Conference on Information Management and Technology (ICIMTech), pp.732-737, 2023.

[2] Shengdang He, Yuanyuan Zou, Derui Wang, "A Review of Vision-Based Hand Action Recognition Techniques", 2023 42nd Chinese Control Conference (CCC), pp.8413-8418, 2023.

[3] M. Karam, A framework for research and design of gesture-based human-computer interactions [D], University of Southampton, 2006.

[4] Y. Golhar and U. Shrawankar, "Human gesture detection & recognition: A need of an era", 2016.

[5] X. Li, "Gesture recognition based on fuzzy c-means clustering algorithm" in Department of Computer Science, The University of Tennessee Knoxville, 2003.

[6] R. Z. Khan and N. A. Ibraheem, "Hand gesture recognition: a literature review", International journal of artificial Intelligence & Applications, vol. 3, no. 4, pp. 161, 2012.

9. APPENDIX (CODE)

```
import cv2
import mediapipe as mp
import time
import numpy as np
from controlkeys import right_pressed,left_pressed,up_pressed,down_pressed
from controlkeys import KeyOn, KeyOff
#import pyautogui

left_key_pressed=left_pressed
right_key_pressed=right_pressed
up_key_pressed=up_pressed
down_key_pressed=down_pressed

time.sleep(2.0)
current_key_pressed = set()

mp_draw=mp.solutions.drawing_utils
mp_hand=mp.solutions.hands

tipIds=[4,8,12,16,20]
```

```
video=cv2.VideoCapture(0)


def get_label(index, hand, results):
    output = None
    for idx, classification in enumerate(results.multi_handedness):
        if classification.classification[0].index == index:
            # Process results
            label = classification.classification[0].label
            score = classification.classification[0].score
            text = '{} {}'.format(label, round(score, 2))

            # Extract Coordinates
            coords = tuple(np.multiply(
                np.array((hand.landmark[mp_hand.HandLandmark.WRIST].x,
hand.landmark[mp_hand.HandLandmark.WRIST].y)),
                [1280, 720]).astype(int))

            output = text, coords

    return output
with mp_hand.Hands(min_detection_confidence=0.5,
               min_tracking_confidence=0.5) as hands:
    while True:
        keyPressed = False
        break_pressed=False
        jump_pressed=False
        dunk_pressed=False
        accelerator_pressed=False
        key_count=0
        key_pressed=0
        ret,image=video.read()
        image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image.flags.writeable=False
        results=hands.process(image)
        image.flags.writeable=True
        image=cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        lmList=[]
        text=''
        if results.multi_hand_landmarks:
            for idx, classification in enumerate(results.multi_handedness):
                if classification.classification[0].index == idx:
                    label = classification.classification[0].label
                    text = '{}'.format(label)
                else:
                    label = classification.classification[0].label
                    text = '{}'.format(label)
            for hand_landmark in results.multi_hand_landmarks:
```

```
                myHands=results.multi_hand_landmarks[0]
                for id, lm in enumerate(myHands.landmark):
                    h,w,c=image.shape
                    cx,cy= int(lm.x*w), int(lm.y*h)
                    lmList.append([id,cx,cy])
                mp_draw.draw_landmarks(image, hand_landmark,
mp_hand.HAND_CONNECTIONS)
        fingers=[]

        if len(lmList)!=0:
            if lmList[tipIds[0]][1] > lmList[tipIds[0]-1][1]:
                fingers.append(1)
            else:
                fingers.append(0)
            for id in range(1,5):
                if lmList[tipIds[id]][2] < lmList[tipIds[id]-2][2]:
                    fingers.append(1)
                else:
                    fingers.append(0)
            total=fingers.count(1)
            if total==4 and text=="Right":
                cv2.rectangle(image, (400, 300), (600, 425), (255, 255, 255),
cv2.FILLED)
                cv2.putText(image, "LEFT", (400, 375),
cv2.FONT_HERSHEY_SIMPLEX,
                    2, (0, 0, 255), 5)
                # cv2.rectangle(image, (100, 300), (200, 425), (255, 255,
255), cv2.FILLED)
                # cv2.putText(image, text, (100, 375),
cv2.FONT_HERSHEY_SIMPLEX,
                #             2, (0, 0, 255), 5)
                KeyOn(left_key_pressed)
                break_pressed=True
                current_key_pressed.add(left_key_pressed)
                key_pressed=left_key_pressed
                keyPressed = True
                key_count=key_count+1
                #pyautogui.press("left")
            elif total==5 and text=="Left":
                cv2.rectangle(image, (400, 300), (600, 425), (255, 255, 255),
cv2.FILLED)
                cv2.putText(image, " RIGHT", (400, 375),
cv2.FONT_HERSHEY_SIMPLEX,
                    2, (0, 255, 0), 5)
                # cv2.rectangle(image, (100, 300), (200, 425), (255, 255,
255), cv2.FILLED)
                # cv2.putText(image, text, (100, 375),
cv2.FONT_HERSHEY_SIMPLEX,
```

```
                #             2, (0, 0, 255), 5)

                KeyOn(right_key_pressed)
                key_pressed=right_key_pressed
                accelerator_pressed=True
                keyPressed = True
                current_key_pressed.add(right_key_pressed)
                key_count=key_count+1
                #pyautogui.press("right")

            elif total==1:
                cv2.rectangle(image, (400, 300), (600, 425), (255, 255, 255),
cv2.FILLED)
                cv2.putText(image, "UP", (400, 375), cv2.FONT_HERSHEY_SIMPLEX,
                    2, (0, 255, 0), 5)

                KeyOn(up_key_pressed)
                key_pressed=up_key_pressed
                jump_pressed=True
                keyPressed = True
                current_key_pressed.add(up_key_pressed)
                key_count=key_count+1
            elif total==0:
                cv2.rectangle(image, (400, 300), (600, 425), (255, 255, 255),
cv2.FILLED)
                cv2.putText(image, "Down", (400, 375),
cv2.FONT_HERSHEY_SIMPLEX,
                    2, (0, 255, 0), 5)

                KeyOn(down_key_pressed)
                key_pressed=down_key_pressed
                down_pressed=True
                keyPressed = True
                current_key_pressed.add(down_key_pressed)
                key_count=key_count+1
            else:
                pass

        if not keyPressed and len(current_key_pressed) != 0:
            for key in current_key_pressed:
                KeyOff(key)
            current_key_pressed = set()
        elif key_count==1 and len(current_key_pressed)==2:
            for key in current_key_pressed:
                if key_pressed!=key:
                    KeyOff(key)
            current_key_pressed = set()
            for key in current_key_pressed:
```

```
                KeyOff(key)
            current_key_pressed = set()


            # if lmList[8][2] < lmList[6][2]:
            #     print("Open")
            # else:
            #     print("Close")
        scale_percent = 500 # percent of the original size
        width = int(image.shape[1] * scale_percent / 100)
        height = int(image.shape[0] * scale_percent / 100)
        dimension = (width, height)
        # resize image
        #resized_image = cv2.resize(image, dimension, interpolation =
cv2.INTER_AREA)
        #cv2.imshow("Frame",resized_image)
        cv2.imshow("Frame",image)
        k=cv2.waitKey(1)
        if k == ord('q'):
            break
video.release()
cv2.destroyAllWindows()
```